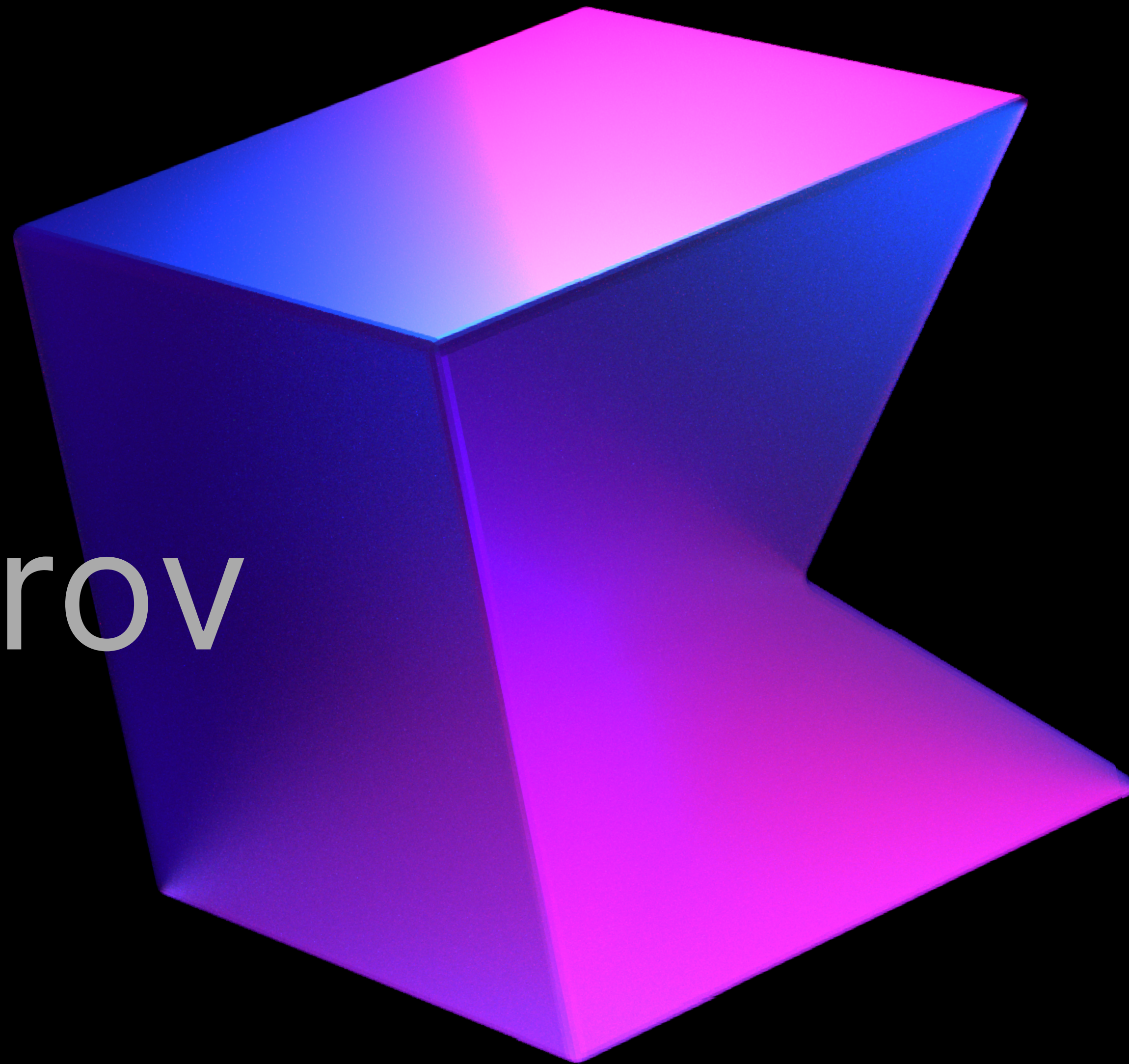


◀ Kotlin 1.4 Online Event

A Look Into the Future

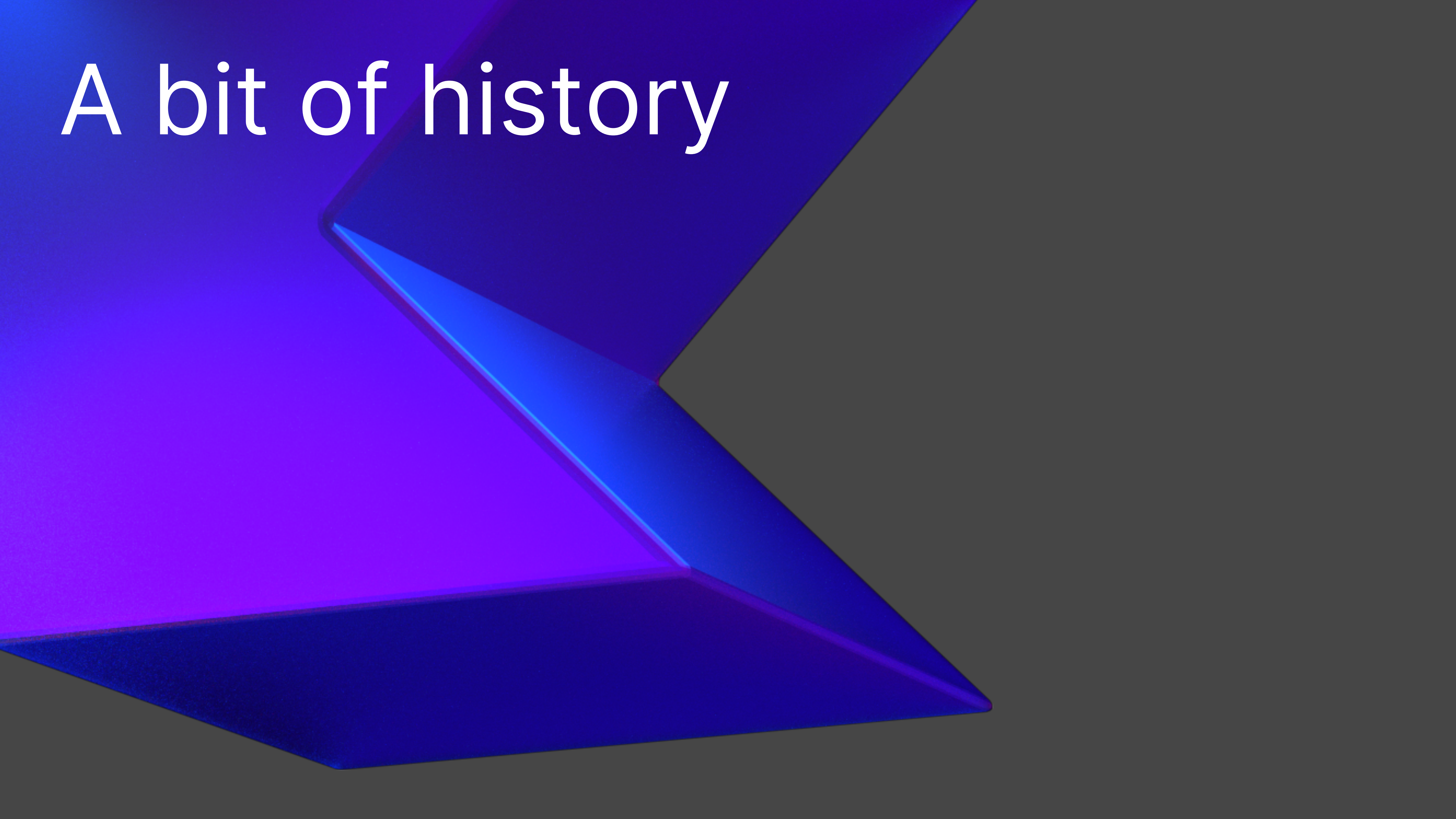
Roman Elizarov



@relizarov

October 12, 2020

A bit of history



Kotlin 1.0



**Programming Language
for JVM and Android**

Posted on February 15, 2016 by Andrey Breslav

A vertical blue line with a downward-pointing arrow at the bottom. A blue arrow-shaped box points to the line from the left, containing the year '2016'.

2016

Kotlin 1.1 Released with JavaScript Support, Coroutines and more



language for JVM, Android & JS

2017

Posted on March 1, 2017 by Roman Belov



Kotlin 1.2 Released: Sharing Code between Platforms



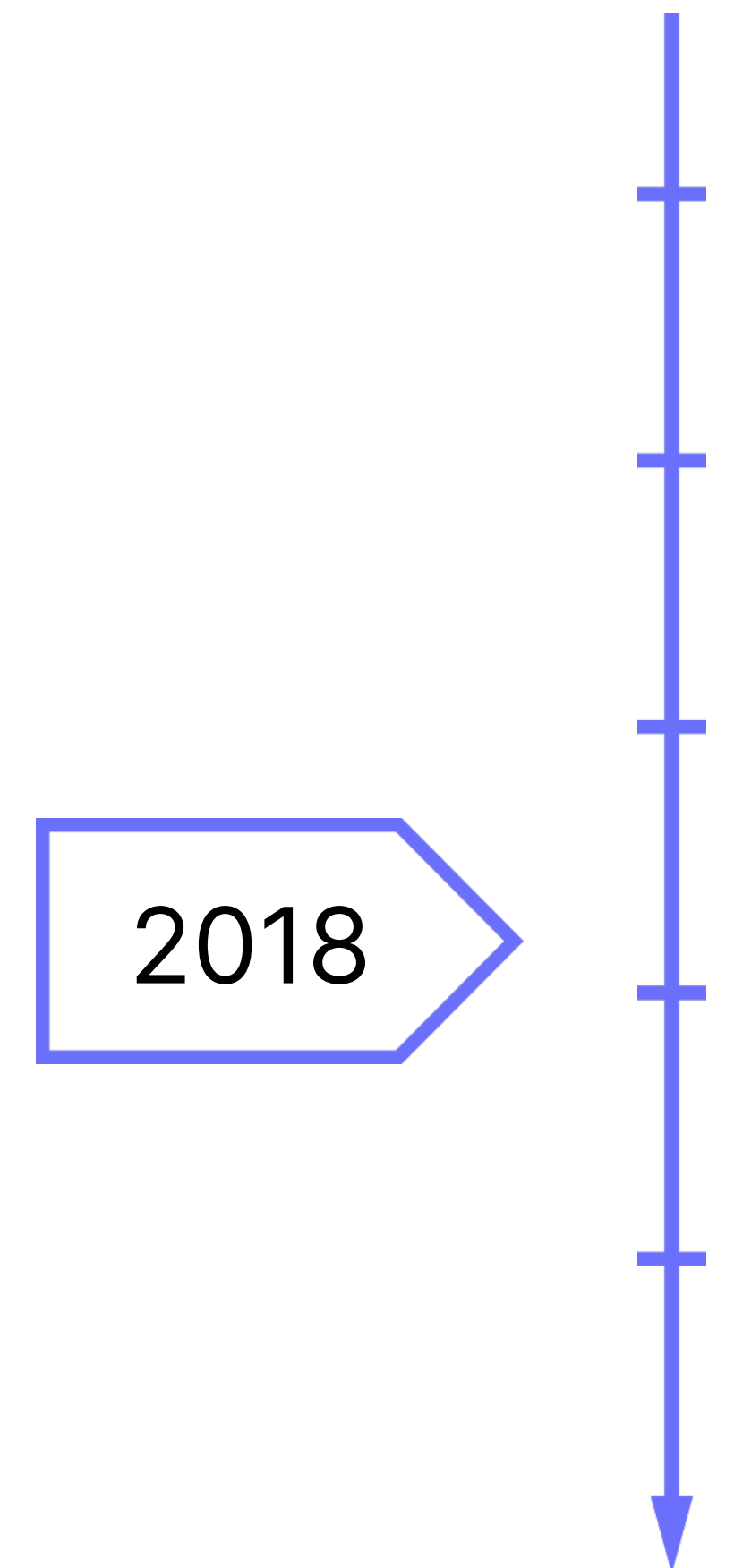
Posted on November 28, 2017 by Dmitry Jemerov

2017

Kotlin 1.3 Released with Coroutines, Kotlin/Native Beta, and more



Posted on October 29, 2018 by Roman Belov



Kotlin 1.4 Released with a Focus on Quality and Performance



Posted on August 17, 2020 by Svetlana Isakova

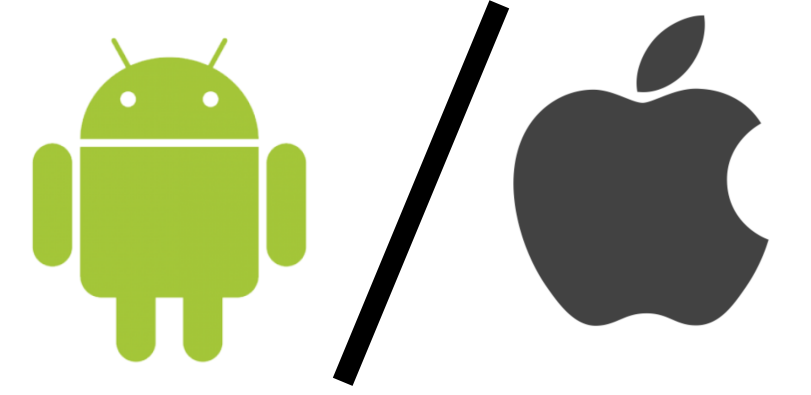
2020



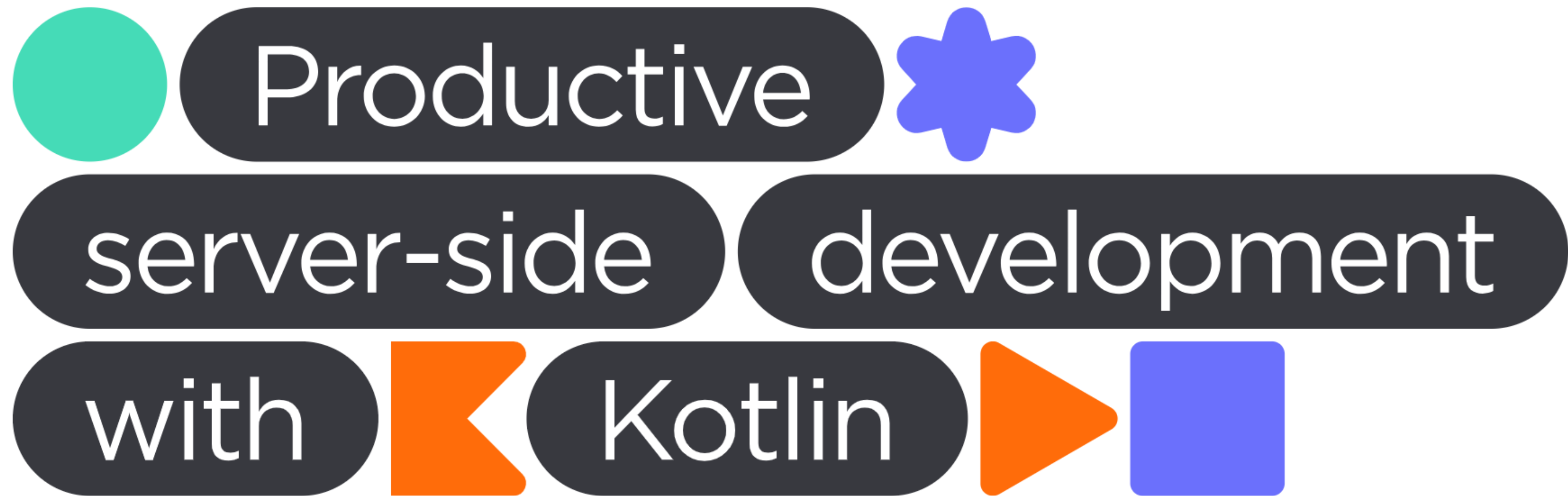
Near future

Plans

Sharing code



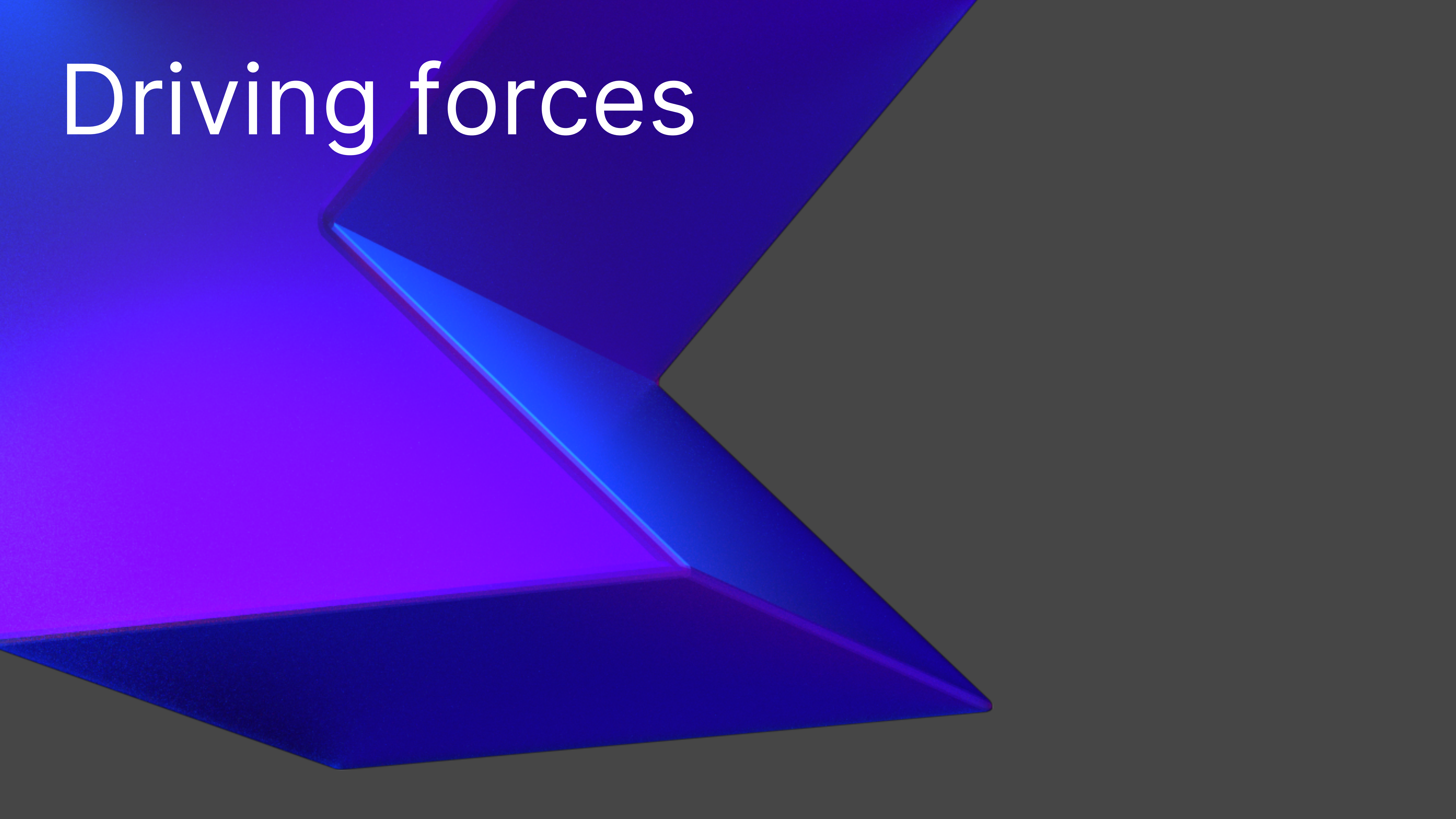
JVM server/interoperability



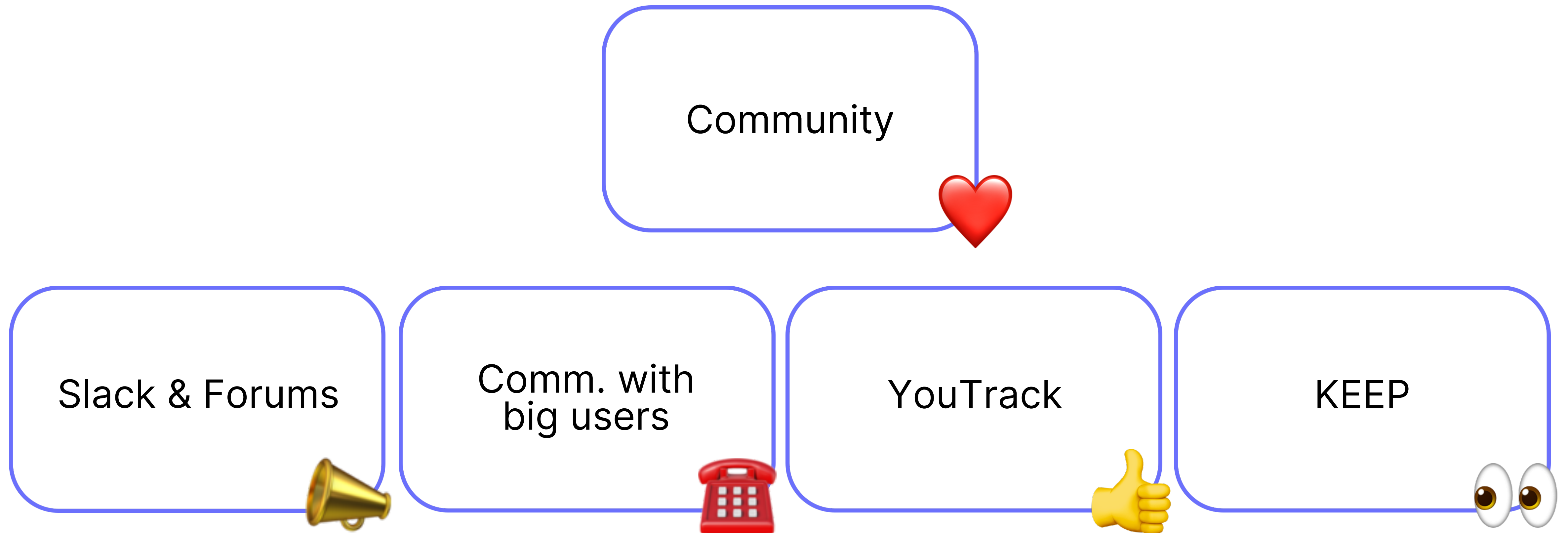
Java interoperability: upcoming

- All new Java APIs: seamless interop
- JEP 359: Records (Preview)
- JEP 384: Records (Second Preview)
- JEP 360: Sealed Classes (Preview)

Driving forces



Driving forces: what to focus on?



YouTrack vs KEEP

- KEEP → design documents
- Worked out and prototyped
- KEEP issues → corrections
- Problems, ideals, proposals → YouTrack `#language design`

<https://kotlin.in/issue>

YouTrack: language design

1.1

~~KT-6947~~ Created by Andreas Sinz 5 years ago Updated by John-Paul Cunliffe 4 years ago Visible to All Users

Callable reference with expression on the left hand side 70 

1.2

~~KT-11235~~ Created by Sébastien Deleuze 5 years ago Updated by Dmitry Konchalenkov 9 months ago Visible to All Users

Allow specifying array annotation attribute single value without arrayOf() 72 

1.3

~~KT-4895~~ Created by Ilya Ryzhenkov 6 years ago Updated by Alexey Belkov 7 months ago Visible to All Users

Support assignment of "when" subject to a variable 129 

1.4

~~KT-7770~~ Created by Sergei Lebedev 5 years ago Updated by Ivan Kubyshkin 2 months ago Visible to All Users

SAM for Kotlin classes 211 

Distant future

Speculative, we are looking for feedback

The most voted request now

KT-11968 Created by [Eric Tsang](#) 4 years ago Updated by [Eduardo Fonseca](#) 2 weeks ago

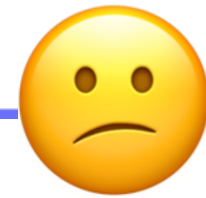
Visible to [All Users](#) ▼

★ Adding statically accessible members to an existing Java class via extensions

479 

KT-11968: Statically accessible extensions

All Kotlin extensions are
resolved statically



KT-11968: Statically accessible extensions

3rd-party type

val Intent.Companion.*SCHEME_SMS*: String get() = "sms"

Companion type

Code with receiver

What are you trying to achieve?

KT-11968: Statically accessible extensions

```
val Intent.Companion.SCHEME_SMS: String get() = "sms"
```

What are you trying to achieve?

Intent.*SCHEME_SMS*

Similar/related problem

```
object Delegates {  
    fun <T : Any> notNull(): ...  
    // other declarations  
}
```

What are you trying to achieve?

Similar/related problem

```
object Delegates {  
    fun <T : Any> notNull(): ...  
    // other declarations  
}
```

What are you trying to achieve?

Delegates.`notNull()`

What is object?

```
object Delegates {  
    fun <T : Any> notNull(): ...  
    // other declarations  
}
```

- Instance

```
val x = Delegates
```

- Type

```
x is Delegates
```

- Namespace

```
Delegates.notNull()
```


What is object?

```
object Delegates {  
    fun <T : Any> notNull(): ...  
    // other declarations  
}
```

- Instance
val x = Delegates
 - Type
x is Delegates
 - Namespace
Delegates.notNull()
- } Library maintenance burden

What if you could declare just a namespace?

```
object Delegates {  
    fun <T : Any> notNull(): ...  
    // other declarations  
}
```

- Namespace
Delegates.notNull()

What if you could declare just a namespace?

```
namespace Delegates {  
    fun <T : Any> notNull(): ...  
    // other declarations  
}
```

- Namespace
Delegates.`notNull()`

Enables: companion namespaces

```
class Example {  
    companion object {  
        private val SOME_CONST = ...  
    }  
}
```

Enables: companion namespaces

```
class Example {  
    namespace {  
        private val SOME_CONST = ...  
    }  
}
```

Enables: namespaces extensions

```
val Intent.Companion.SCHEME_SMS: String get() = "sms"
```


Enables: namespaces extensions

```
val namespace<Intent>.SCHEME_SMS: String get() = "sms"
```

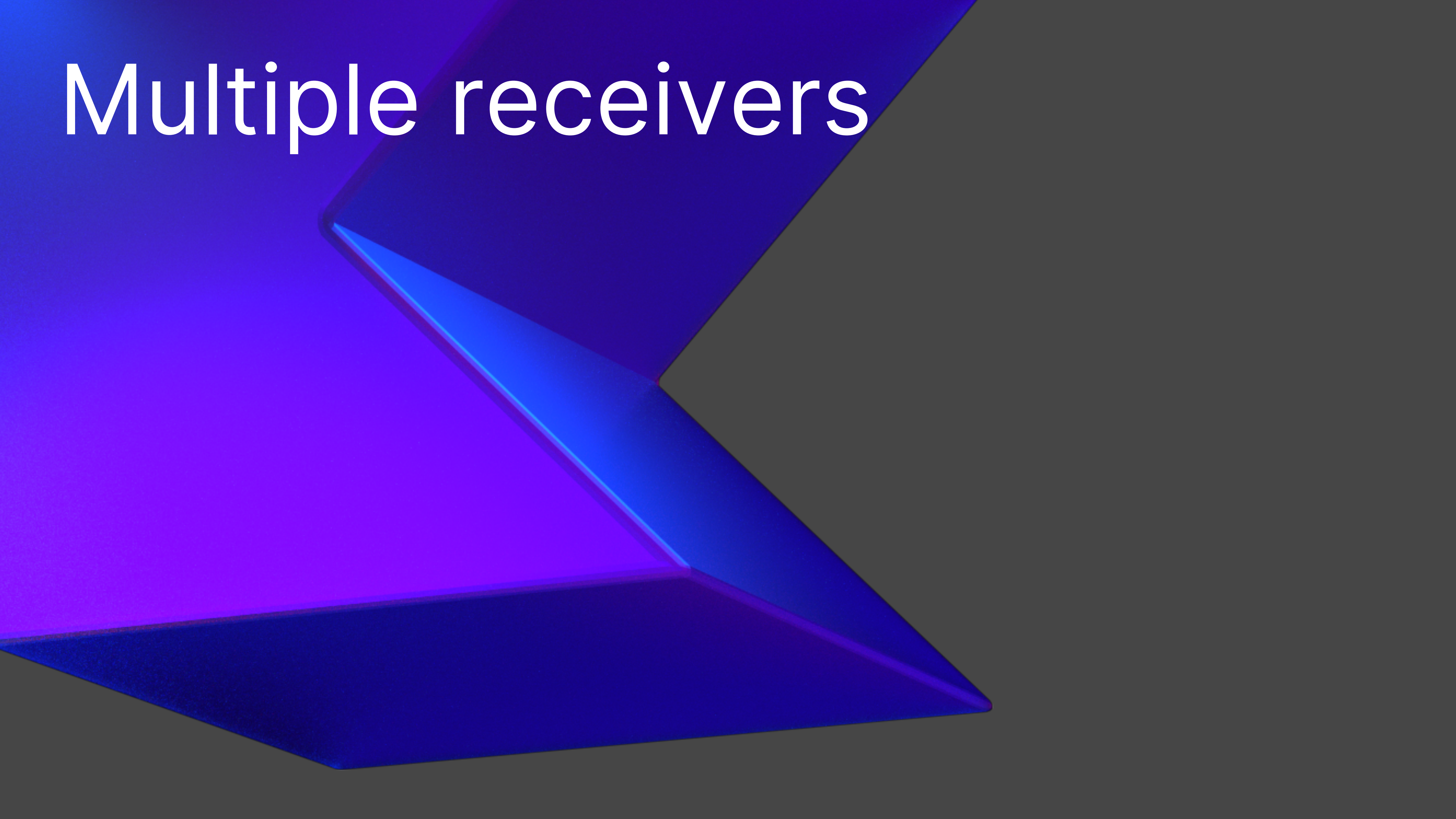
Code without receiver

Intent.*SCHEME_SMS*



What
we wanted!

Multiple receivers



KT-10468 Created by [Damian Wieczorek](#) 5 years ago Updated by [Margarita Bobova](#) 4 weeks ago

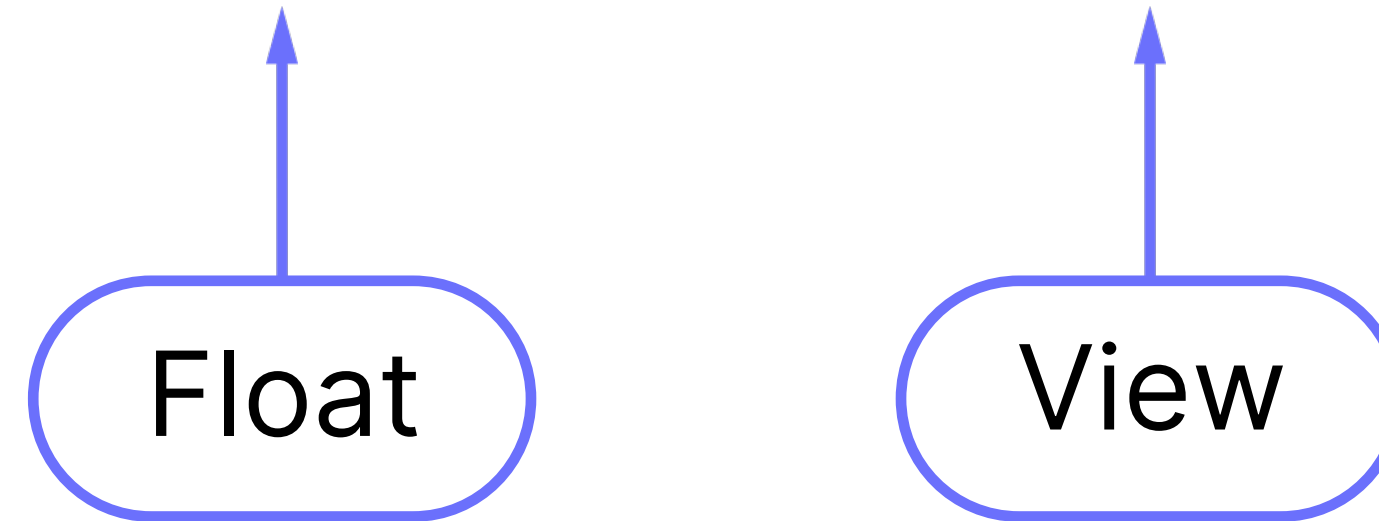
Visible to [All Users](#) ▼

★ Multiple receivers on extension functions/properties

81 

Member extensions

```
class View {  
    fun Float.dp() = this * resources.displayMetrics.density  
}
```



KT-10468: Multiple receivers

```
fun (View, Float).dp() = this * resources.displayMetrics.density
```

KT-10468: Multiple receivers

```
fun View.Float.dp() = ...
```


KT-10468: Multiple receivers

```
fun Float.dp(implicit view: View) = ...
```

Syntactic analogy

```
with(view) {  
    42f.dp()  
}
```

Syntactic analogy

with<View>

```
fun Float.dp() = this * resources.displayMetrics.density
```


Take it further

```
inline fun <T> withTransaction(block: () -> T): T {  
    val tx = beginTransaction()  
    return try {  
        block()  
    } finally {  
        tx.commit()  
    }  
}
```

Take it further

```
inline fun <T> withTransaction(block: () -> T): T {  
    val tx = beginTransaction()  
    return try {  
        block()  
    } finally {  
        tx.commit()  
    }  
}
```

Take it further

```
inline fun <T> withTransaction(block: () -> T): T {  
    val tx = beginTransaction()  
    return try {  
        block()  
    } finally {  
        tx.commit()  
    }  
}
```

Take it further

```
inline fun <T> withTransaction(block: () -> T): T {  
    val tx = beginTransaction()  
    return try {  
        block()  
    } finally {  
        tx.commit()  
    }  
}
```

```
fun doSomething() {  
    withTransaction {  
        // code  
    }  
}
```



No magic

Decorators


```
inline decorator fun <T> withTransaction(block: () -> T): T {  
    val tx = beginTransaction()  
    return try {  
        block()  
    } finally {  
        tx.commit()  
    }  
}
```

```
fun doSomething() {  
    withTransaction {  
        // code  
    }  
}
```

Decorators

```
inline decorator fun <T> withTransaction(block: () -> T): T {  
    val tx = beginTransaction()  
    return try {  
        block()  
    } finally {  
        tx.commit()  
    }  
}
```

```
@withTransaction  
fun doSomething() {  
    // code  
}
```



The best
of two
worlds

Decorators with receivers

```
inline decorator fun <T> Tx.withTransaction(block: () -> T): T {  
    begin()  
    return try {  
        block()  
    } finally {  
        commit()  
    }  
}
```

```
@withTransaction  
fun doSomething() {  
    // code  
}
```


Gets additional receiver Tx



Decorators with receivers

`@with<View>`

```
fun Float.dp() = this * resources.displayMetrics.density
```



Just a
standard
decorator!

Public/private property types

It does not have to be complicated

KT-14663 Created by Svetlana Isakova 4 years ago Updated by David Friehs 6 months ago

Visible to All Users ▼

★ Support having a "public" and a "private" type for the same property

119 👍

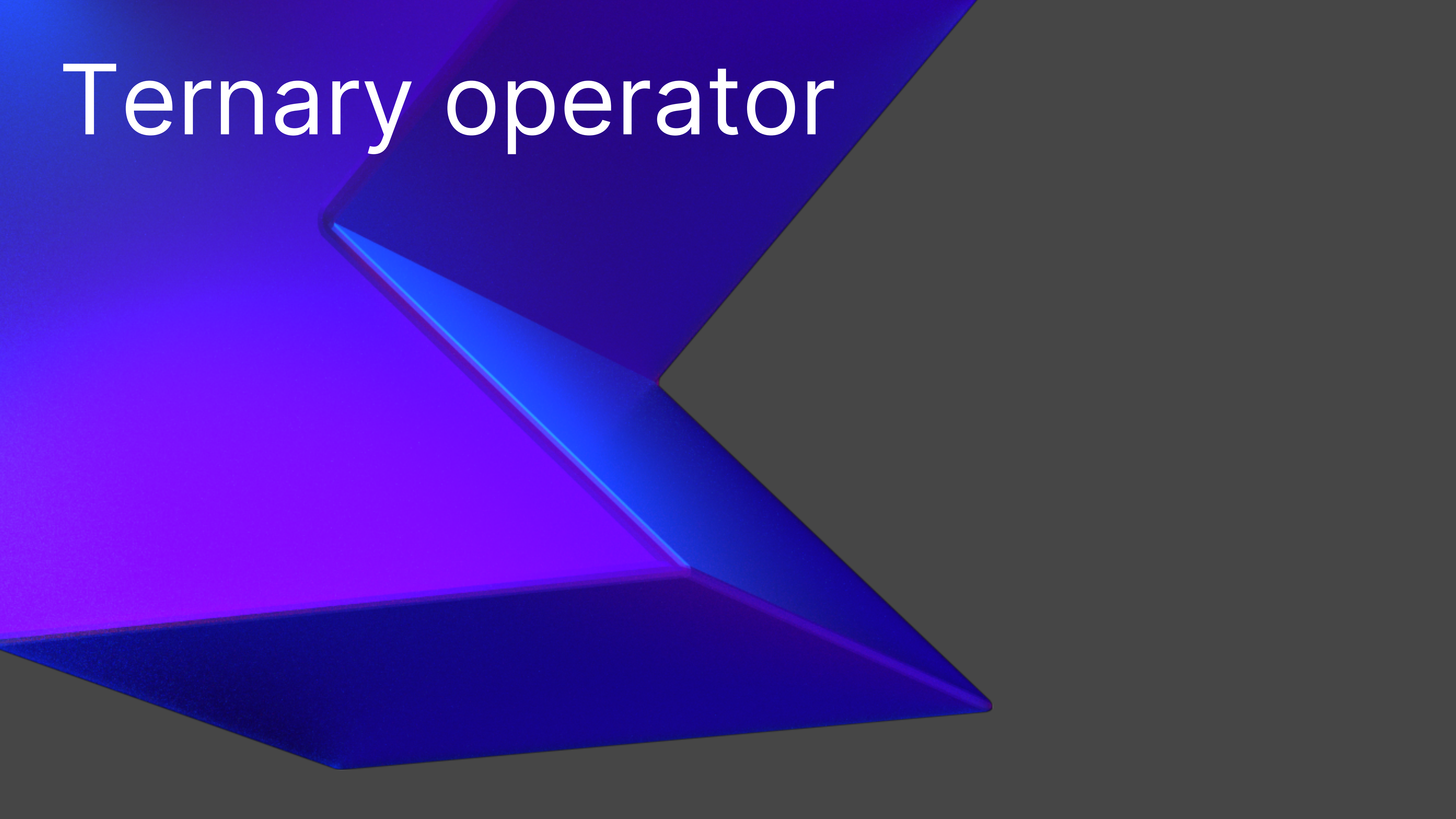
```
private val items = mutableListOf<Item>()  
    public get(): List<Item>
```

```
private val _items = mutableListOf<Item>()  
public val items: List<Item> get() = _items
```



Minimal
design
needed

Ternary operator



☆ Support ternary conditional operator 'foo ? a : b'

80 👍

- Kotlin has “if” expression

```
if (foo) a else b
```

When to use which

Hard for existing code

- Kotlin consistently uses “?” in the context of nullability

```
foo ?: b
```

Hard for novices, inconsistent

- Boolean abuse in APIs

The goal of Kotlin is to enable type-safe APIs

Do you write nullable or Boolean before ?

Declined

Immutability

Cross-cutting trend

Mutable data

```
data class State(  
    var lastUpdate: Instant,  
    var tags: List<String>  
)
```

| Declare

```
state.lastUpdate = now()  
state.tags += tag
```

| Update

```
notifyOnChange(state.copy())
```

| Share

Immutable data

```
data class State(  
    val lastUpdate: Instant,  
    val tags: List<String>  
)
```

| Declare

Immutable data

```
data class State(  
    val lastUpdate: Instant,  
    val tags: List<String>  
)
```

| Declare

```
state = state.copy(  
    lastUpdate = now(),  
    tag = state.tags + tag  
)
```

| Update

```
notifyOnChange(state)
```

| Share

Can we have cake and eat it, too?

```
val class State(  
    val lastUpdate: Instant,  
    val tags: List<String>  
)
```

| Declare

Value-based class



No stable
identity


Can we have cake and eat it, too?

```
val class State(  
    val lastUpdate: Instant,  
    val tags: List<String>  
)
```

| Declare

```
state.lastUpdate = now()  
state.tags += tag
```

| Update



Copying
syntax
sugar

Can we have cake and eat it, too?

```
val class State(  
    val lastUpdate: Instant,  
    val tags: List<String>  
)
```

| Declare

```
state.lastUpdate = now()  
state.tags += tag
```

| Update

```
notifyOnChange(state)
```

| Share

Experimental inline classes

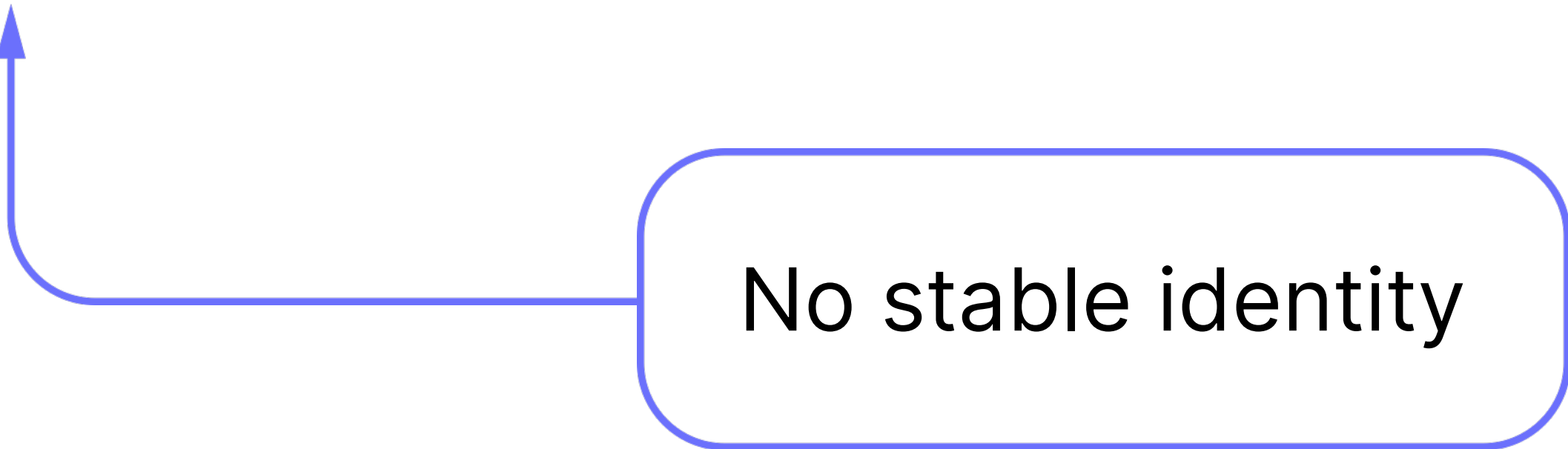
```
inline class Color(val rgb: Int)
```

Confusing with Valhalla inline

A blue rounded rectangular box containing the text "Confusing with Valhalla inline". A blue line extends from the left side of the box, turns upwards, and ends in an arrowhead pointing to the "inline" keyword in the code snippet above.

Stable future for experimental inline classes

```
@__TBD__  
val class Color(val rgb: Int)
```



No stable identity

Stable future for experimental inline classes

Optimize away boxes when possible

@__TBD__

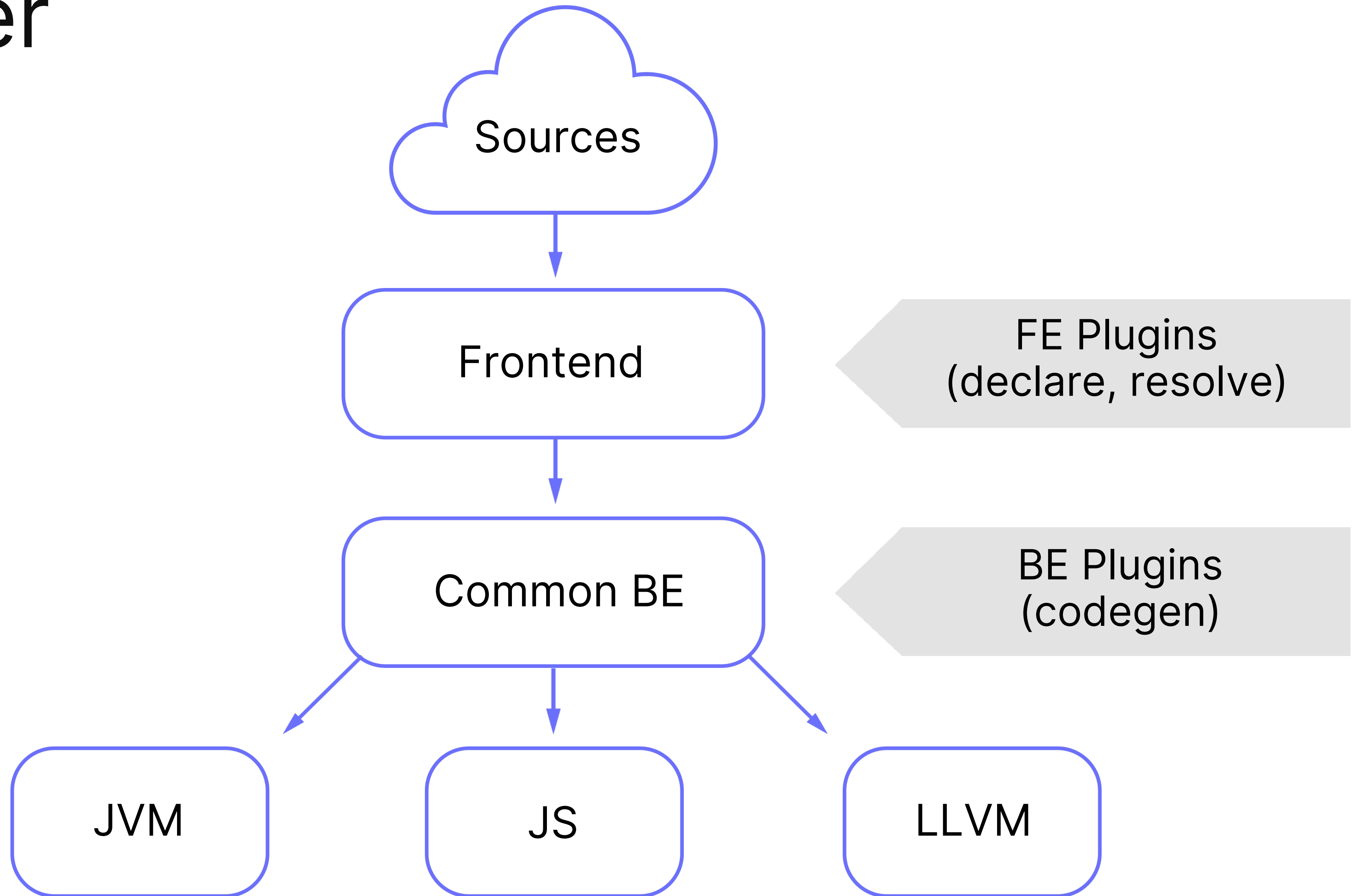
```
val class Color(val rgb: Int)
```

No stable identity

Other contributions to Kotlin features

The background of the slide is an abstract composition of geometric shapes. On the left, there are several overlapping planes in shades of blue and purple, creating a sense of depth and perspective. These planes appear to be part of a larger, three-dimensional structure. To the right of these colored planes is a solid, dark gray area that occupies the right half of the slide. The overall aesthetic is modern and technical, fitting for a presentation about programming language features.

Compiler



JetPack Compose



```
@Composable
fun Greeting(name: String) {
    Surface(color = Color.Yellow) {
        Text(text = "Hello $name!")
    }
}
```

JetPack Compose



@Composable

```
fun Greeting(name: String) {  
    Surface(color = Color.Yellow) {  
        Text(text = "Hello $name!")  
    }  
}
```

A language
feature

Differentiable programming @Facebook

@Differentiable

```
fun foo(x: Float, y: Float): Float {  
    val a = x * y  
    val b = a + 5f  
    val c = b * b * b  
    return c  
}
```



Automatic
differentiation

Growing community

- Arrow KT by 47 Degrees

<https://github.com/arrow-kt/arrow>

- Power asserts by Brian Norman

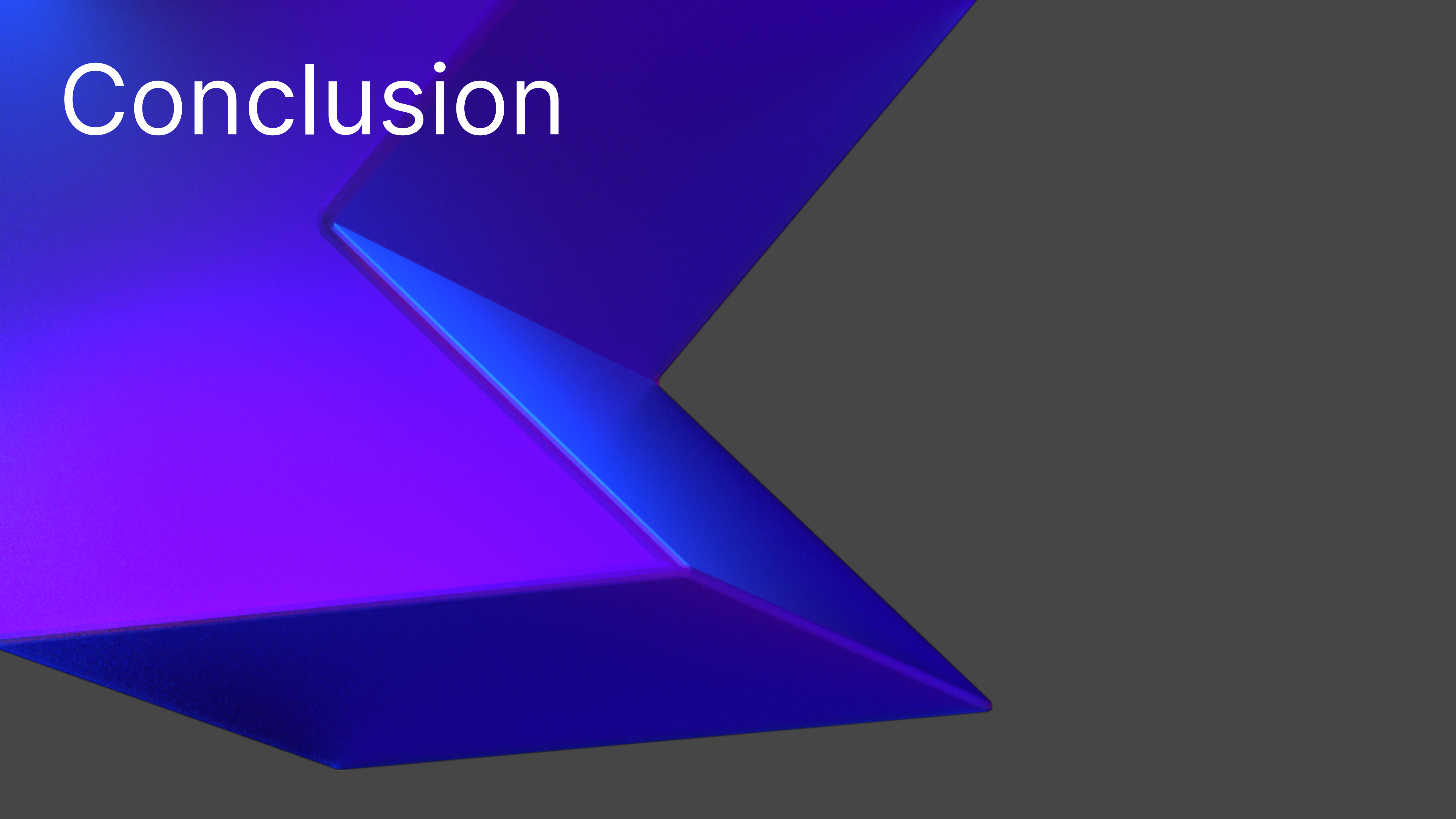
<https://github.com/bnorm/kotlin-power-assert>

- <your project here>

Growing community

- Arrow KT by 47 Degrees
<https://github.com/arrow-kt/arrow>
- Power asserts by Brian Norman
<https://github.com/bnorm/kotlin-power-assert>

Conclusion



Recap

- JVM interop commitment
- Namespaces and extensions
- Multiple receivers
- Public/private property types
- Ternary operator
- Immutability and inline classes
- Other contributions

What else we are looking at?

- More concise syntax for algebraic types
- Data literals (collection literals, tuples, etc)
- Even more flexible properties
- Better API evolution/maintenance facilities
- Constant evaluation/folding
- And more!

Thanks!
Have a nice Kotlin

